US 20050240806A1

(19) **United States**

(12) **Patent Application Publication**

(10) **Pub. No.: US 2005/0240806 A1**

(43) **Pub. Date: Oct. 27, 2005**

Bruckert et al.

(54) **DIAGNOSTIC MEMORY DUMP METHOD IN A REDUNDANT PROCESSOR**

(75) Inventors: **William F. Bruckert**, Los Gatos, CA (US); **James S. Klecka**, Georgetown, TX (US); **James R. Smullen**, Carmel, CA (US)

Correspondence Address:
HEWLETT PACKARD COMPANY
P O BOX 272400, 3404 E. HARMONY ROAD
INTELLECTUAL PROPERTY ADMINISTRATION
FORT COLLINS, CO 80527-2400 (US)

(73) Assignee: **Hewlett-Packard Development Company, L.P.**, Houston, TX

(21) Appl. No.: **10/953,242**

(22) Filed: **Sep. 28, 2004**

Related U.S. Application Data

(60) Provisional application No. 60/557,812, filed on Mar. 30, 2004.

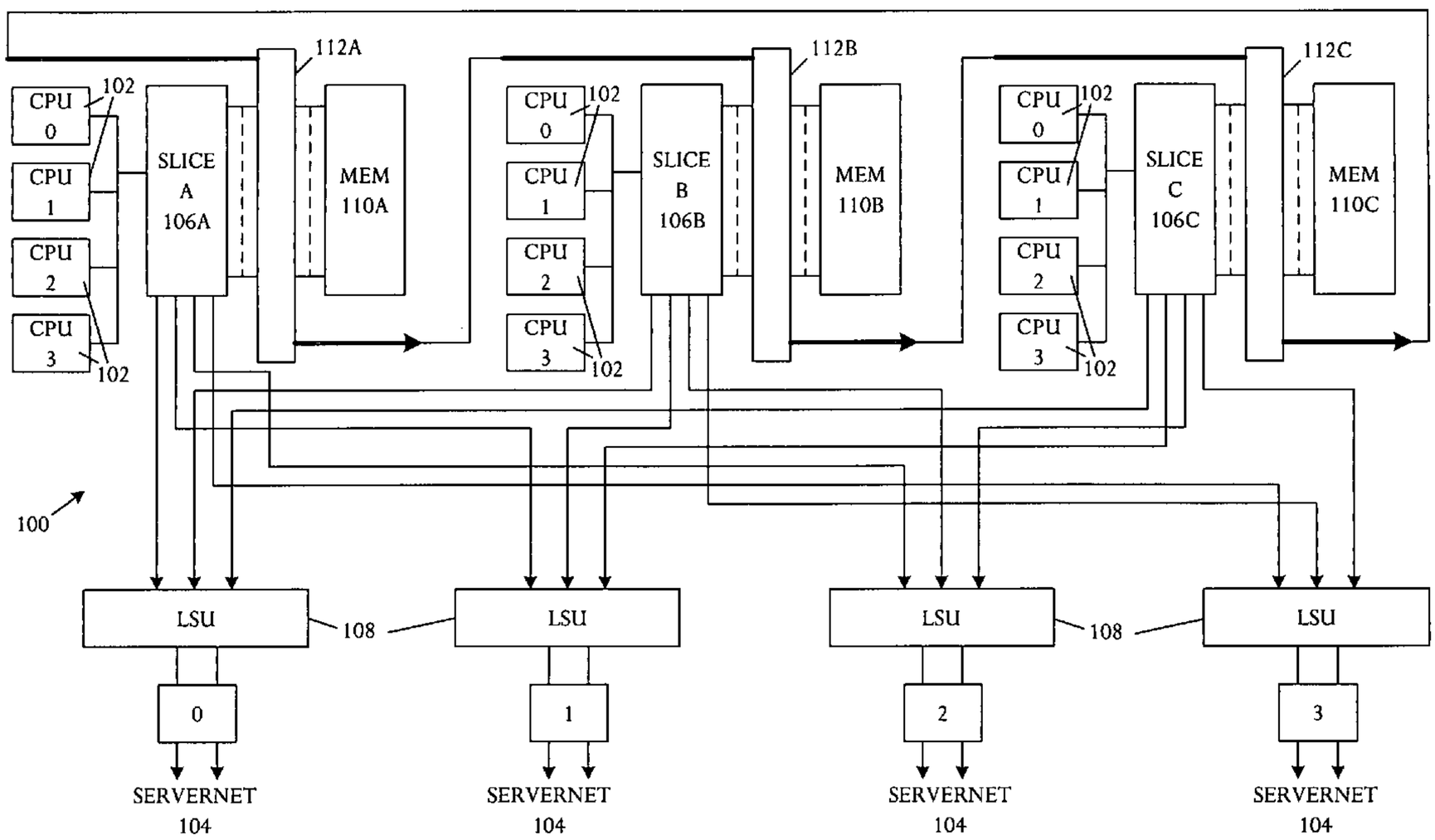
Publication Classification

(51) **Int. Cl.**[7] **G06F 11/00**

(52) **U.S. Cl.** **714/6**; 714/718

(57) **ABSTRACT**

A plurality of redundant, loosely-coupled processor elements are operational as a logical processor. A logic detects a halt condition of the logical processor and, in response to the halt condition, reintegrates and commences operation in less than all of the processor elements leaving at least one processor element nonoperational. The logic also buffers data from the nonoperational processor element in the reloaded operational processor elements and writes the buffered data to storage for analysis.



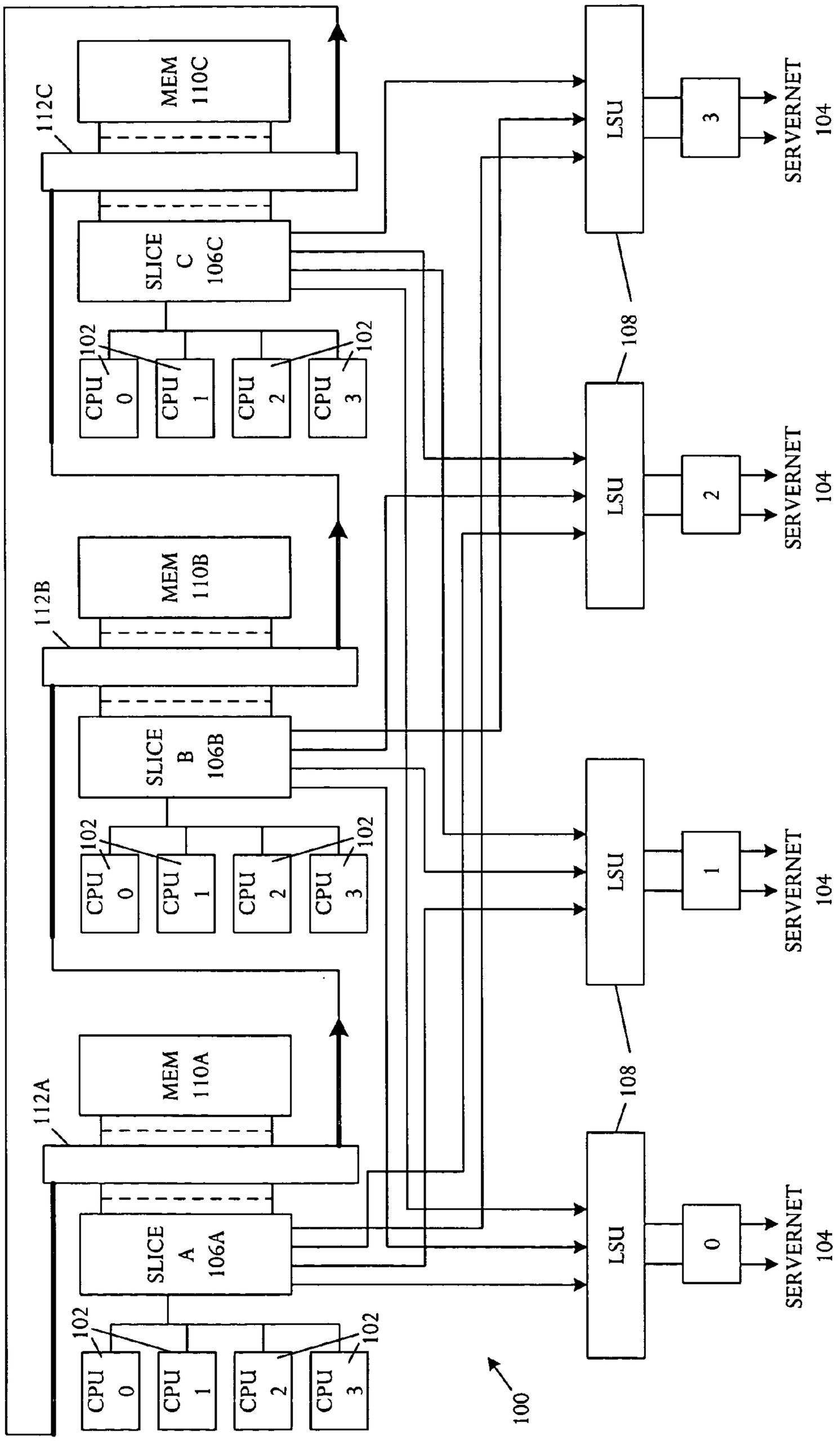

MEM 110C
112C
SLICE C 106C
CPU 0
CPU 1
CPU 2
CPU 3
102
MEM 110B
112B
SLICE B 106B
MEM 110A
112A
SLICE A 106A
LSU
108
3
2
1
0
SERVERNET 104
100

FIG. 1

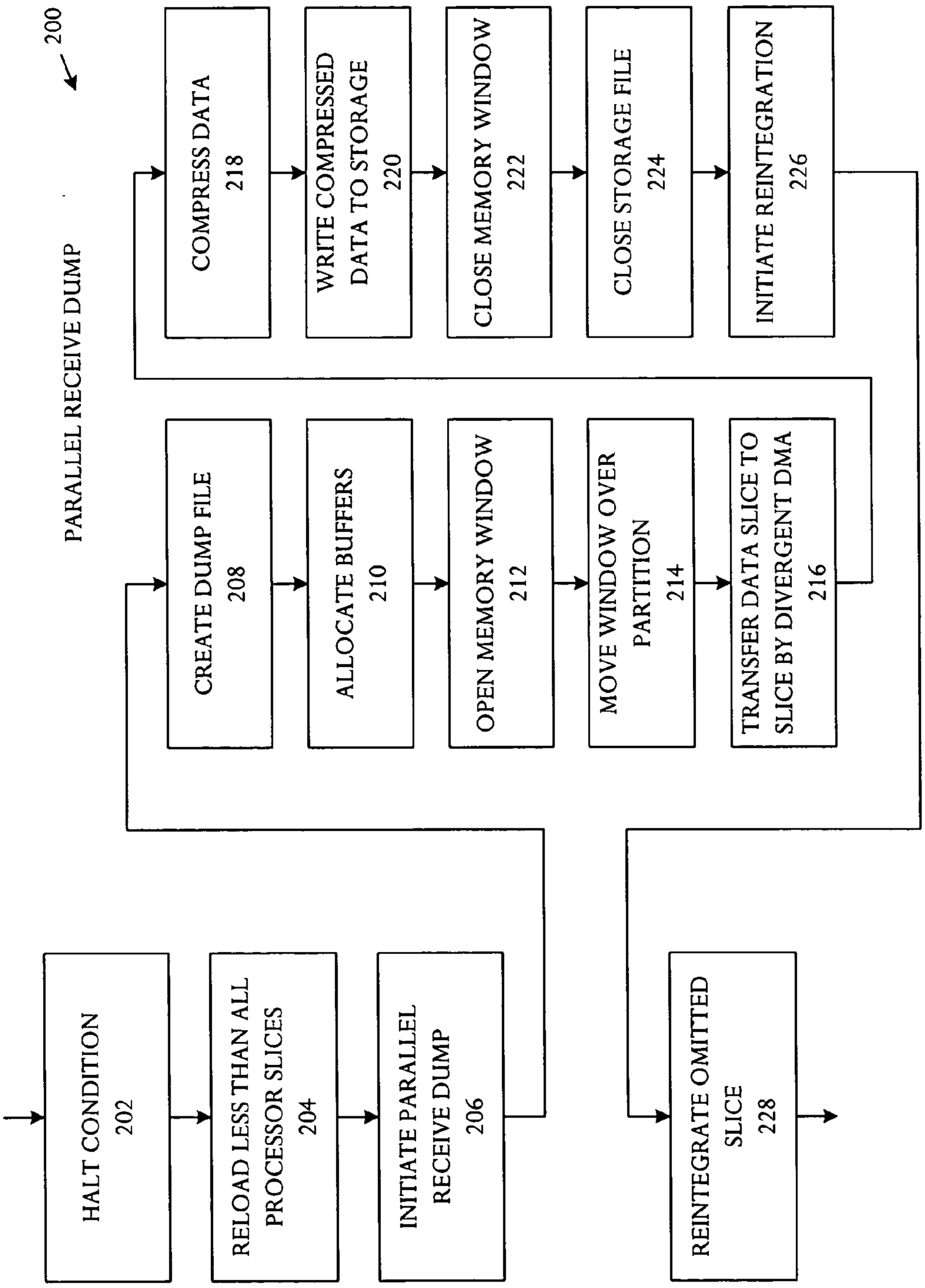

200
PARALLEL RECEIVE DUMP
HALT CONDITION
202
RELOAD LESS THAN ALL PROCESSOR SLICES
204
INITIATE PARALLEL RECEIVE DUMP
206
CREATE DUMP FILE
208
ALLOCATE BUFFERS
210
OPEN MEMORY WINDOW
212
MOVE WINDOW OVER PARTITION
214
TRANSFER DATA SLICE TO SLICE BY DIVERGENT DMA
216
COMPRESS DATA
218
WRITE COMPRESSED DATA TO STORAGE
220
CLOSE MEMORY WINDOW
222
CLOSE STORAGE FILE
224
INITIATE REINTEGRATION
226
REINTEGRATE OMITTED SLICE
228

FIG. 2

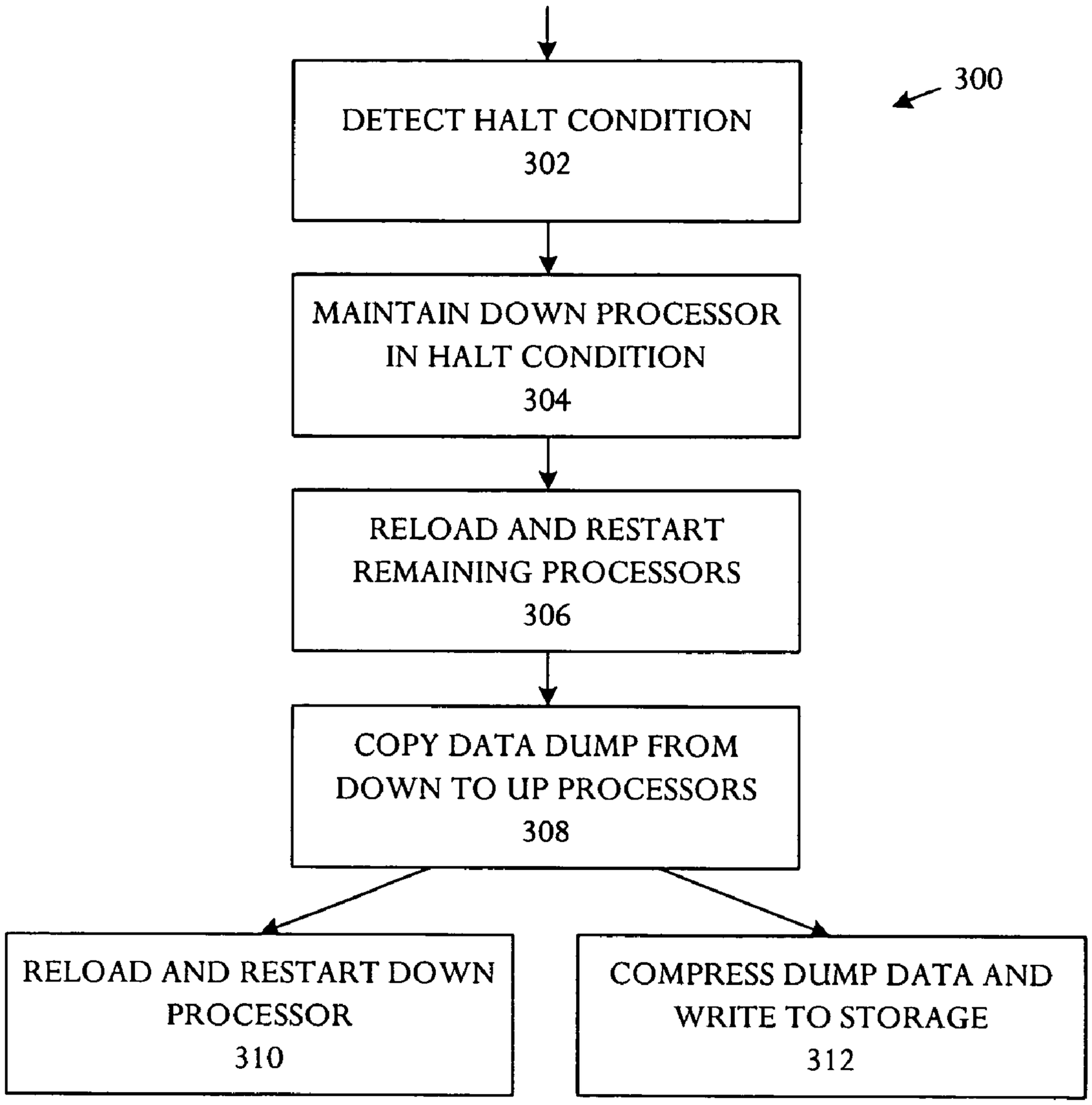

300
DETECT HALT CONDITION
302
MAINTAIN DOWN PROCESSOR IN HALT CONDITION
304
RELOAD AND RESTART REMAINING PROCESSORS
306
COPY DATA DUMP FROM DOWN TO UP PROCESSORS
308
RELOAD AND RESTART DOWN PROCESSOR
310
COMPRESS DUMP DATA AND WRITE TO STORAGE
312

FIG. 3

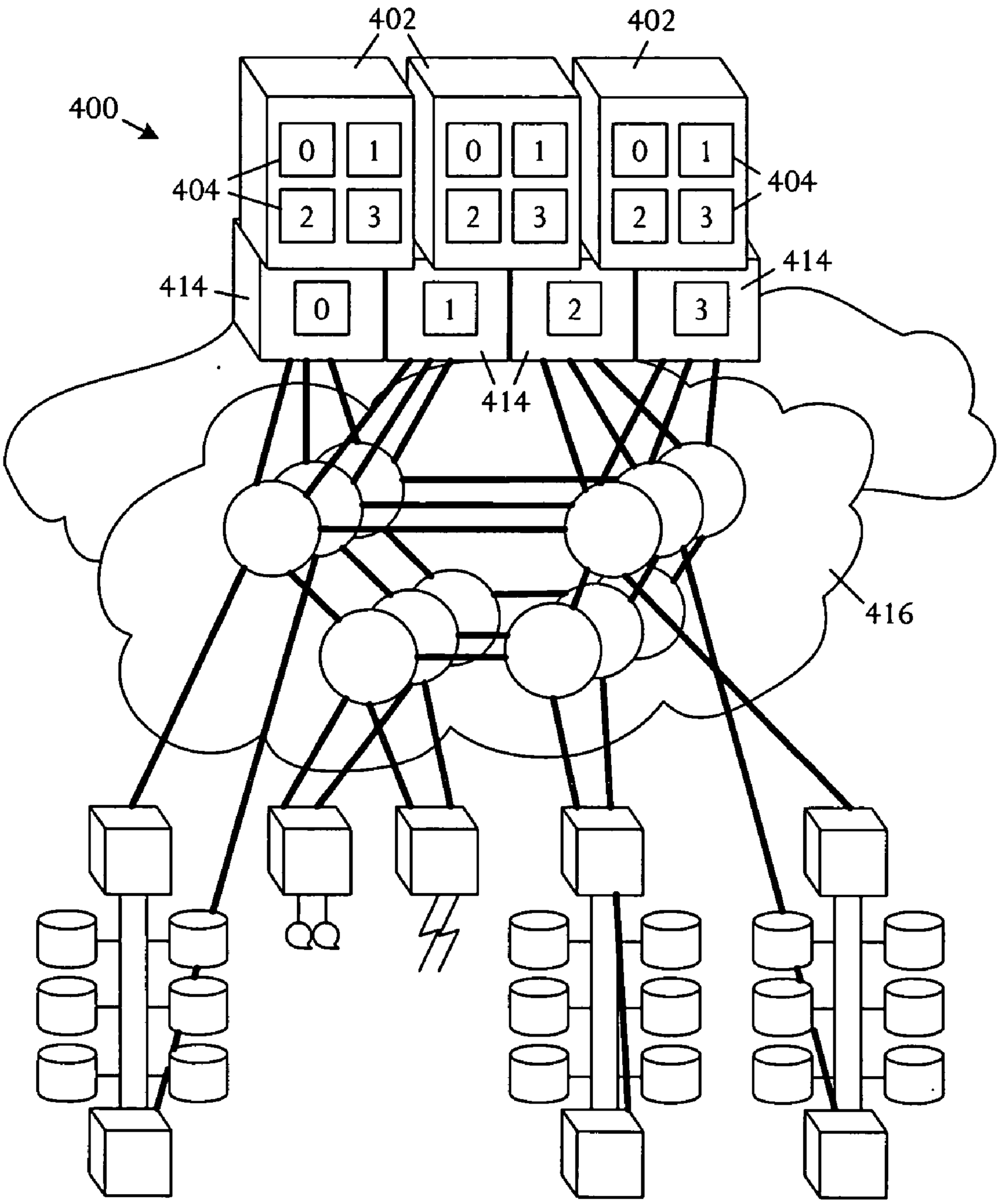


FIG. 4A

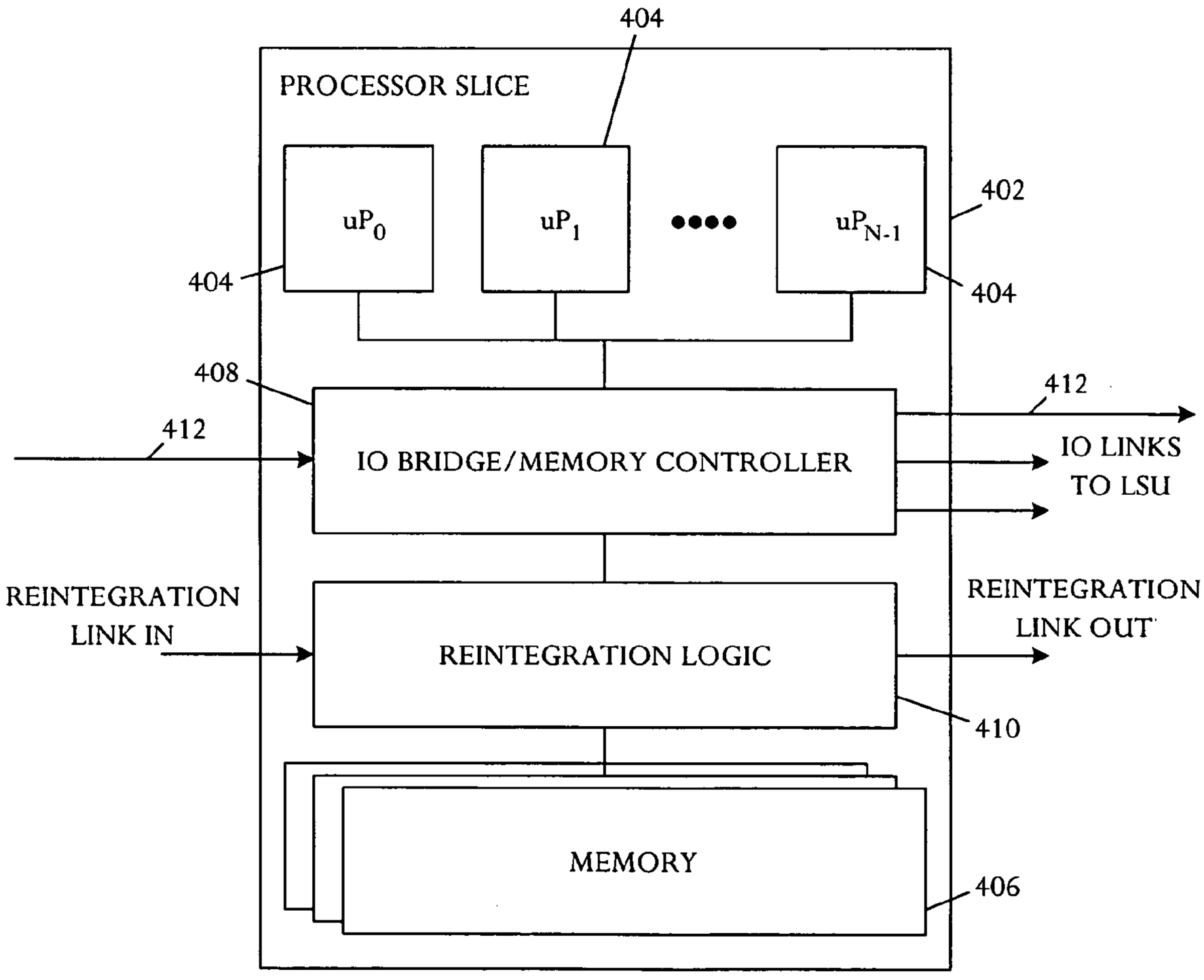

404
PROCESSOR SLICE
uP0
uP1
uPN-1
402
404
404
408
412
412
IO BRIDGE/MEMORY CONTROLLER
IO LINKS
TO LSU
REINTEGRATION
LINK IN
REINTEGRATION LOGIC
REINTEGRATION
LINK OUT
410
MEMORY
406

FIG. 4B

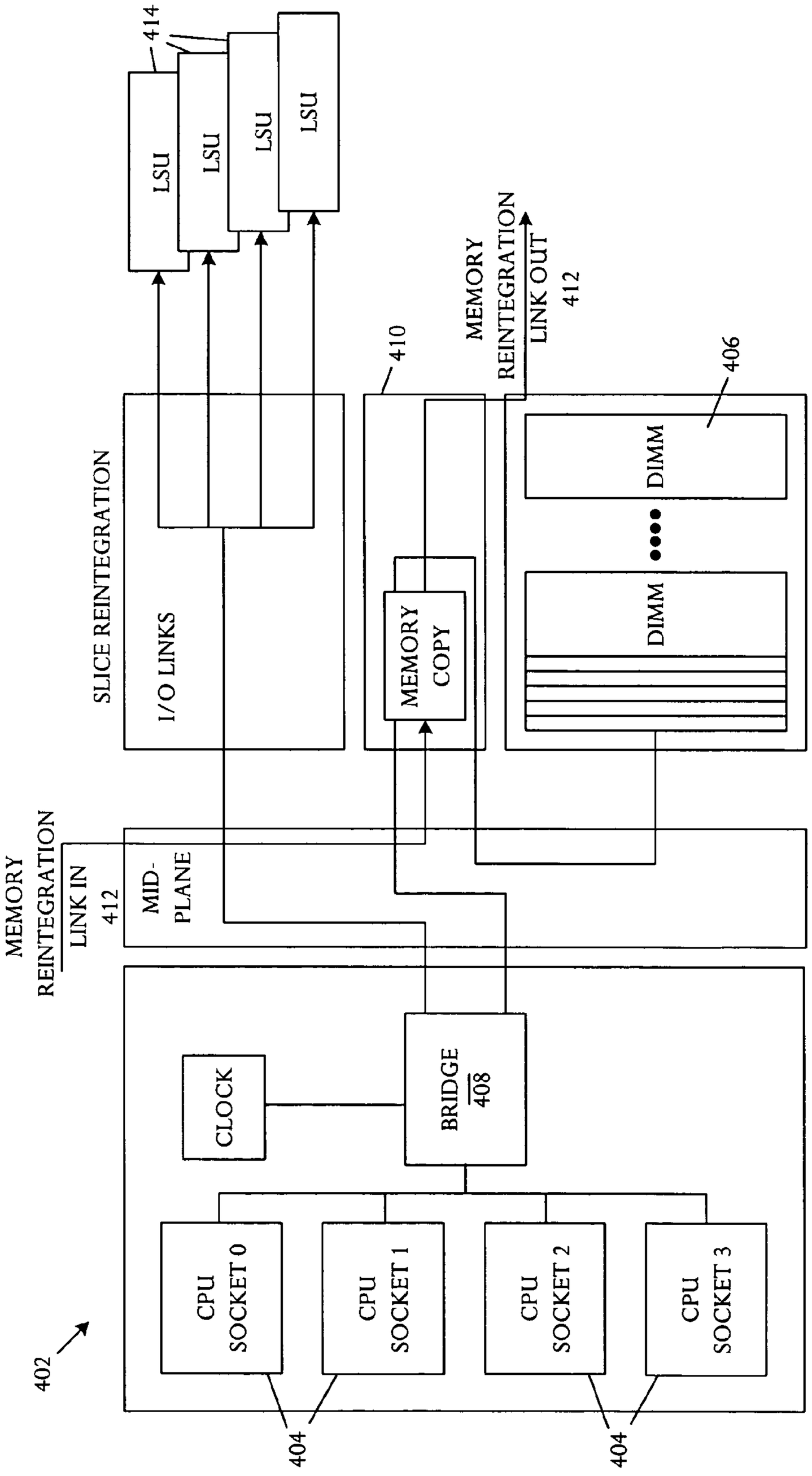

402
CPU SOCKET 0
CPU SOCKET 1
CPU SOCKET 2
CPU SOCKET 3
404
404
CLOCK
BRIDGE 408
MEMORY REINTEGRATION LINK IN 412
MID-PLANE
SLICE REINTEGRATION I/O LINKS
LSU
LSU
LSU
LSU
414
410
MEMORY COPY
MEMORY REINTEGRATION LINK OUT 412
406
DIMM
DIMM

FIG. 4C

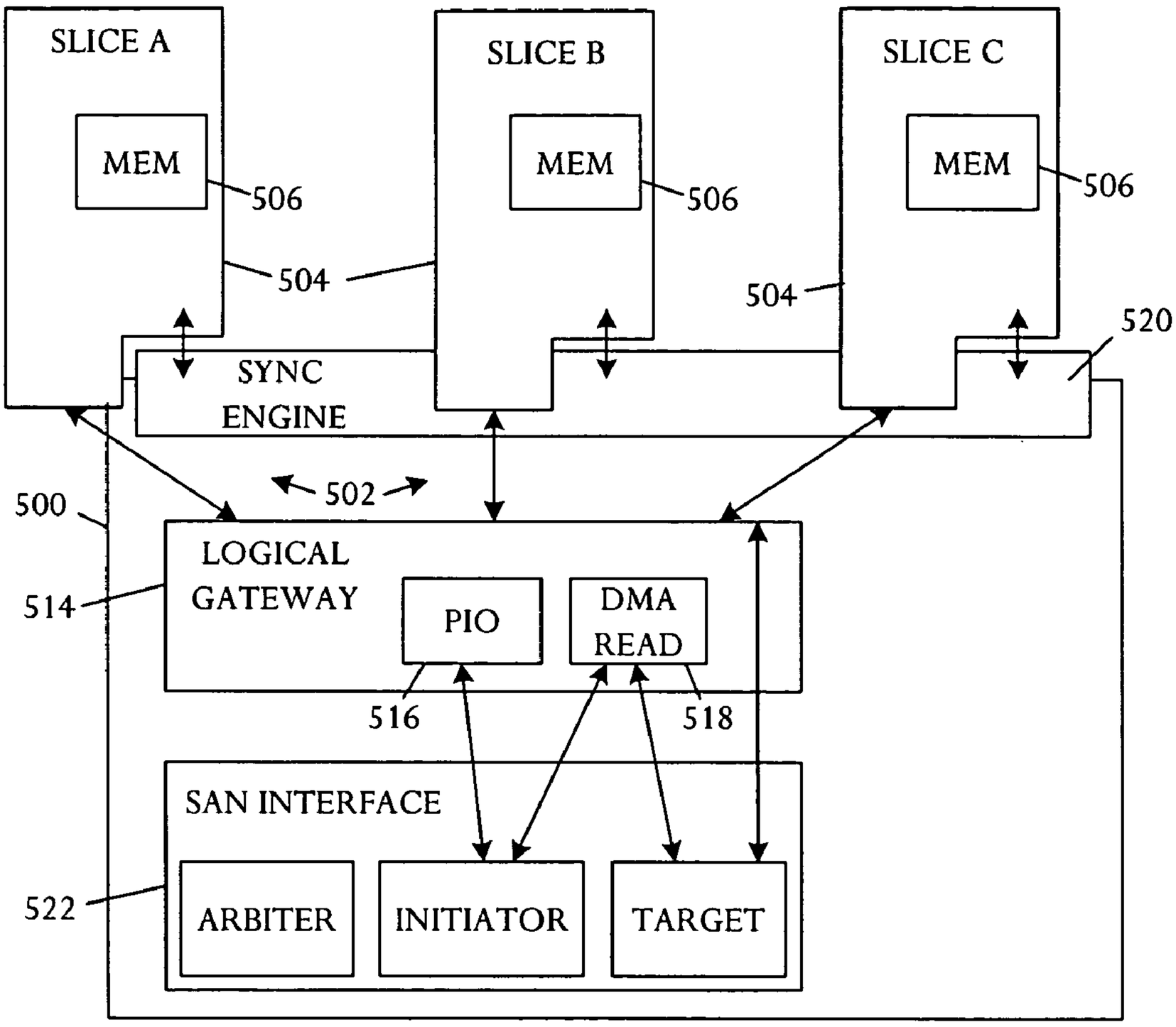

SLICE A
MEM
506
SLICE B
MEM
506
SLICE C
MEM
506
504
504
520
SYNC
ENGINE
502
500
LOGICAL
GATEWAY
514
PIO
DMA
READ
516
518
SAN INTERFACE
522
ARBITER
INITIATOR
TARGET

FIG. 5

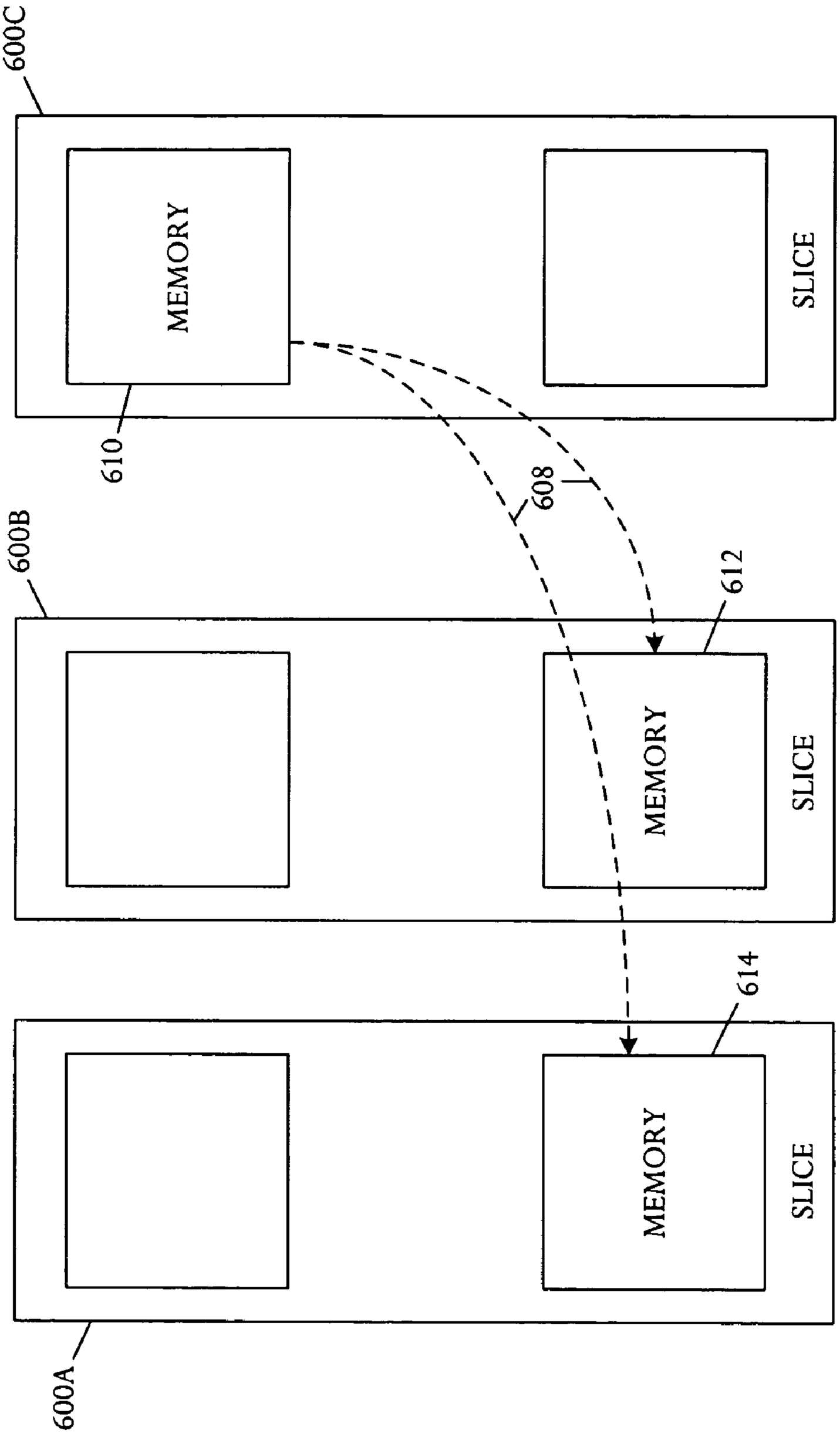

600C
MEMORY
SLICE
610
608
600B
612
MEMORY
SLICE
614
MEMORY
SLICE
600A

FIG. 6

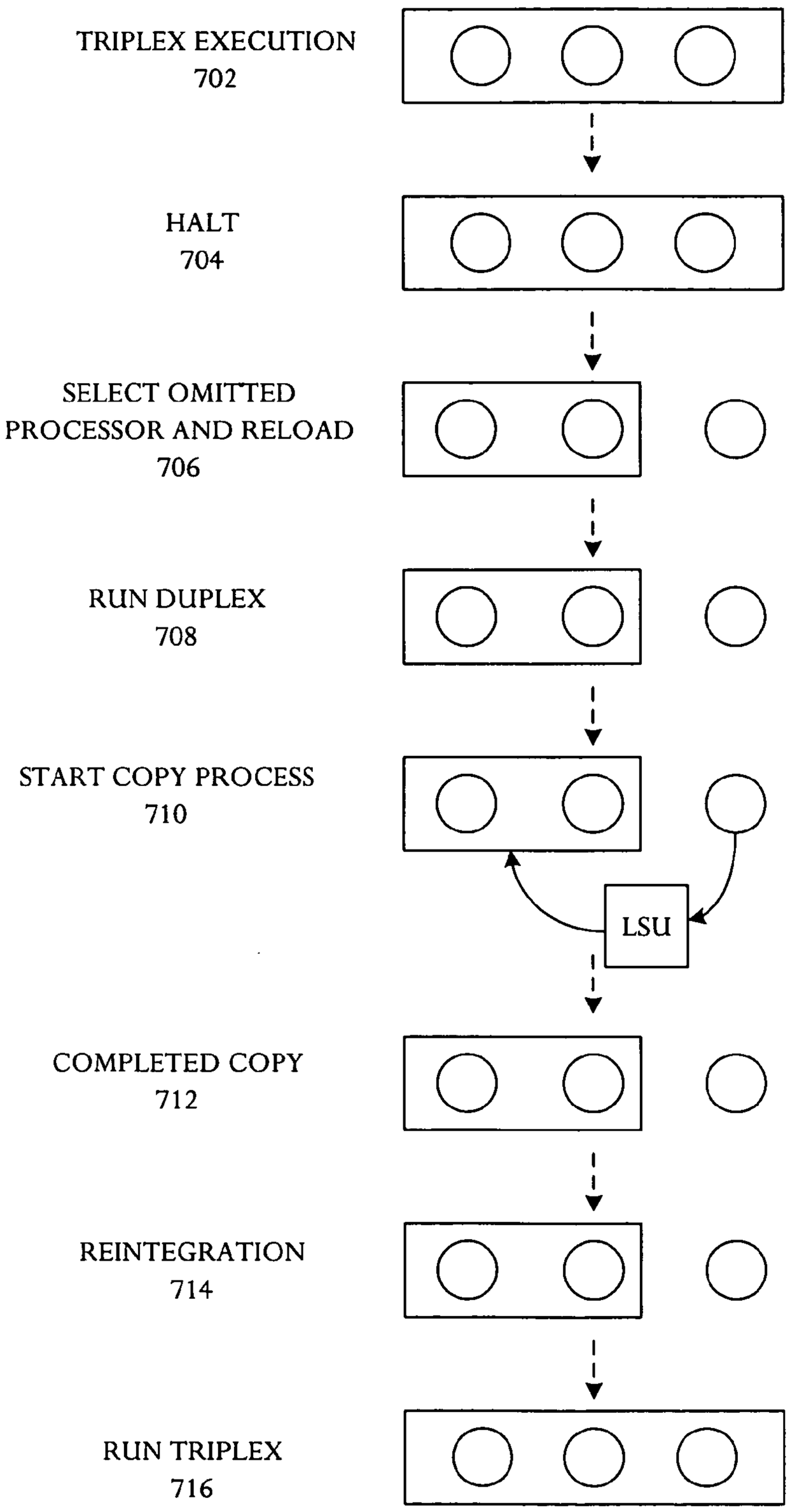

TRIPLEX EXECUTION
702
HALT
704
SELECT OMITTED
PROCESSOR AND RELOAD
706
RUN DUPLEX
708
START COPY PROCESS
710
LSU
COMPLETED COPY
712
REINTEGRATION
714
RUN TRIPLEX
716

FIG. 7

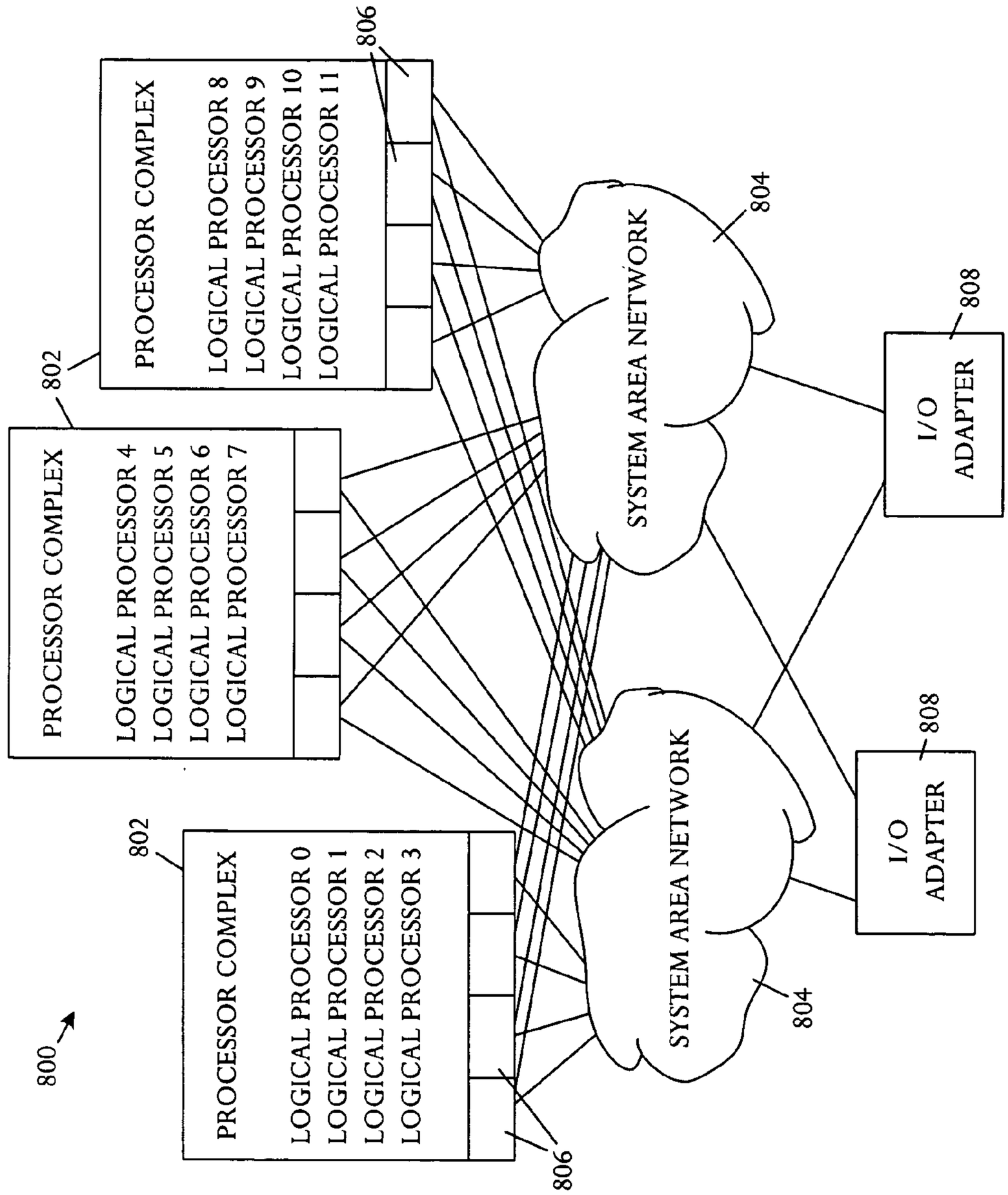

800
802
PROCESSOR COMPLEX
LOGICAL PROCESSOR 0
LOGICAL PROCESSOR 1
LOGICAL PROCESSOR 2
LOGICAL PROCESSOR 3
802
PROCESSOR COMPLEX
LOGICAL PROCESSOR 4
LOGICAL PROCESSOR 5
LOGICAL PROCESSOR 6
LOGICAL PROCESSOR 7
806
804
SYSTEM AREA NETWORK
804
SYSTEM AREA NETWORK
808
I/O ADAPTER
808
I/O ADAPTER

FIG. 8

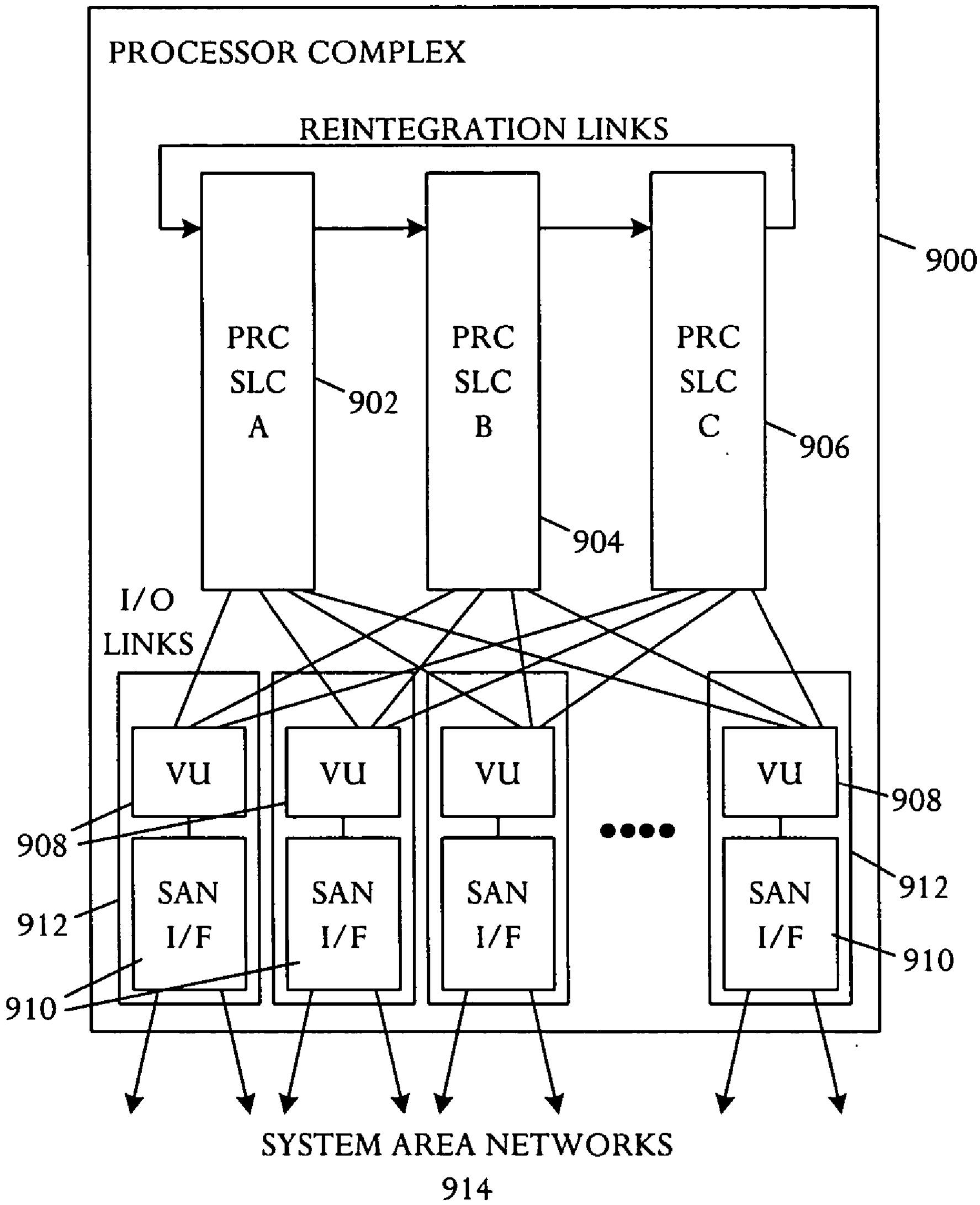

PROCESSOR COMPLEX
REINTEGRATION LINKS
PRC SLC A
PRC SLC B
PRC SLC C
902
904
906
900
I/O LINKS
VU
908
SAN I/F
912
910
SYSTEM AREA NETWORKS
914

FIG. 9

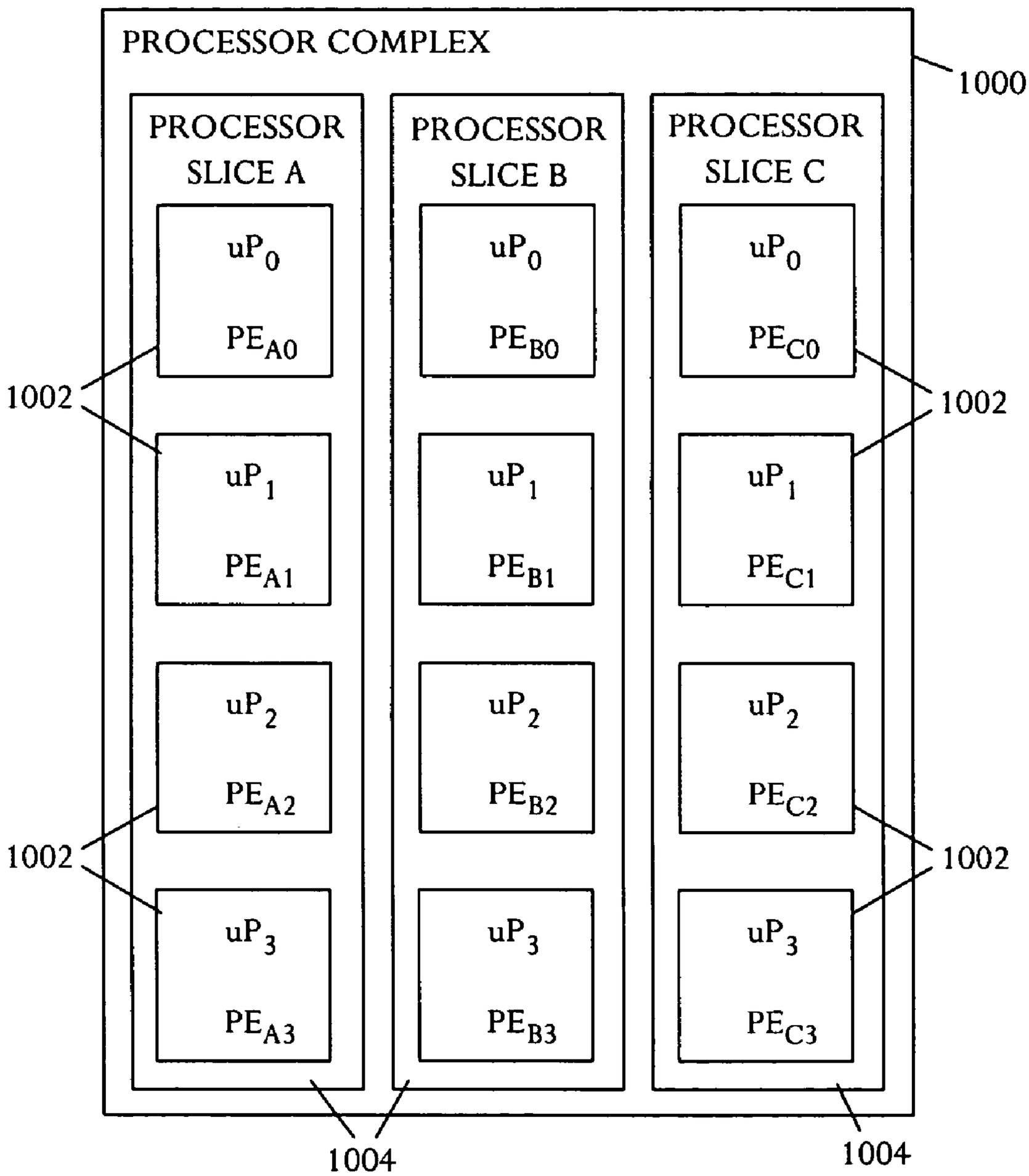

PROCESSOR COMPLEX
1000
PROCESSOR SLICE A
PROCESSOR SLICE B
PROCESSOR SLICE C
uP0
PEA0
PEB0
PEC0
uP1
PEA1
PEB1
PEC1
uP2
PEA2
PEB2
PEC2
uP3
PEA3
PEB3
PEC3
1002
1004

FIG. 10A

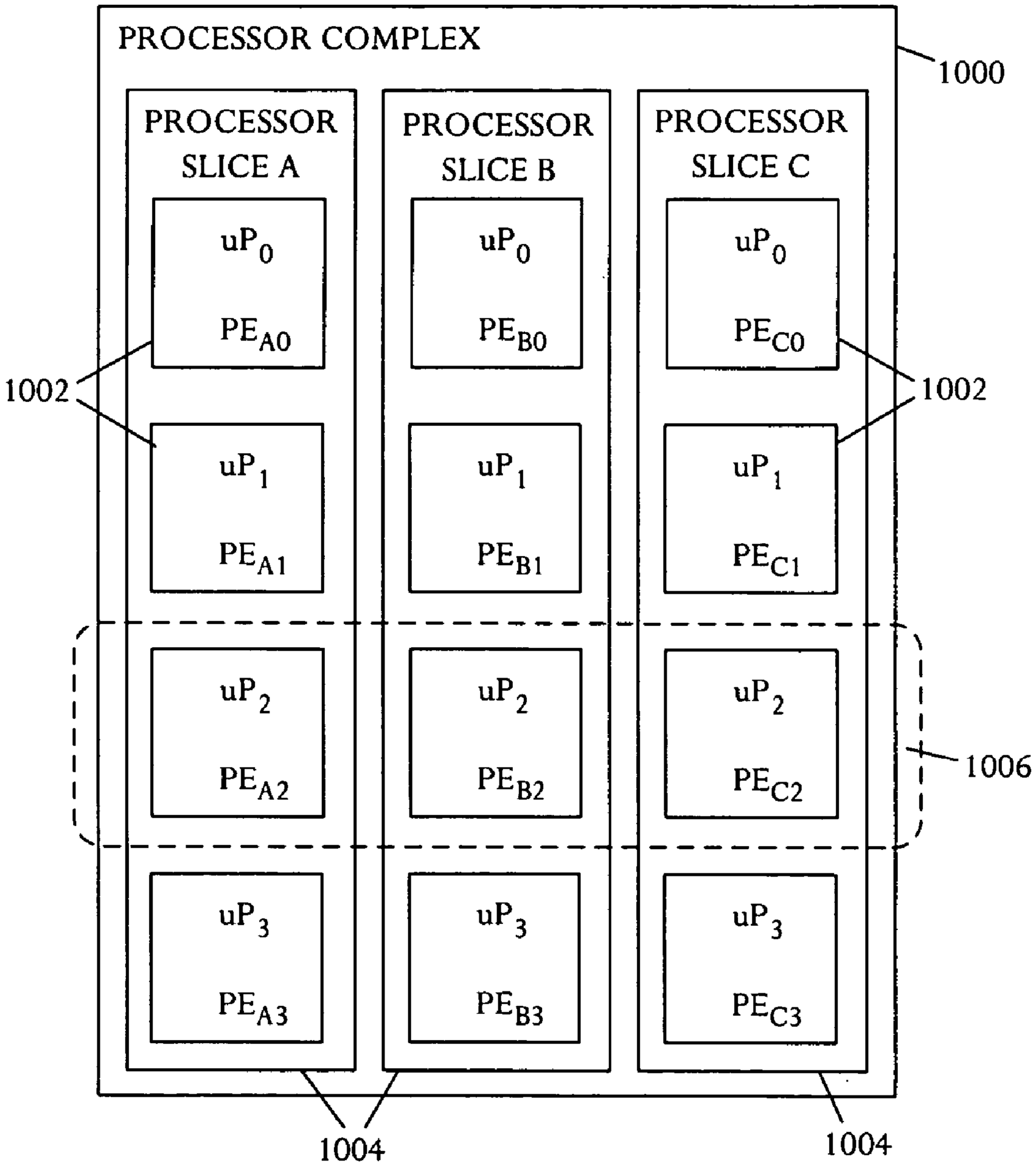

PROCESSOR COMPLEX
1000
PROCESSOR SLICE A
PROCESSOR SLICE B
PROCESSOR SLICE C
uP0
PEA0
PEB0
PEC0
1002
1002
uP1
PEA1
PEB1
PEC1
uP2
PEA2
PEB2
PEC2
1006
uP3
PEA3
PEB3
PEC3
1004
1004

FIG. 10B

DIAGNOSTIC MEMORY DUMP METHOD IN A REDUNDANT PROCESSOR

BACKGROUND OF THE INVENTION

[0001] System availability, scalability, and data integrity are fundamental characteristics of enterprise systems. A nonstop performance capability is imposed in financial, communication, and other fields that use enterprise systems for applications such as stock exchange transaction handling, credit and debit card systems, telephone networks, and the like. Highly reliable systems are often implemented in applications with high financial or human costs, in circumstances of massive scaling, and in conditions that outages and data corruption cannot be tolerated.

[0002] In a continuously available system, a user expects and desires end-to-end application availability, a capability to achieve a desired operation within an acceptable response time. A scalable system is expected to achieve near 100% linear scalability, scaling from a few processors to thousands in a manner that the amount of useful work done when the nth processor is added to a cluster is essentially the same as the incremental amount of work accomplished when the second, third, or additional processors are added.

[0003] One aspect of a highly available and reliable system is a capability to analyze failure events, enabling treatment and possible prevention of such failure conditions. A useful tool for diagnosing system difficulties in a computing system is a memory dump, an output file generated by an operating system during a failure for usage in determining the cause of the failure.

[0004] Unfortunately, capture, handling, and storage of diagnostic information all can impose on performance and availability of a system. For a sophisticated, complex, or large system, acquisition and handling of diagnostic information can last for many minutes or possibly hours, thereby compromising system availability.

[0005] Various techniques are used to reduce time expended in writing debugging information and limit the storage space for storing a crash dump file in the event of a fatal error. For example, the stored debug information may be reduced to cover only the operating system or kernel level memory, allowing analysis of nearly all kernel-level system errors. Unfortunately the kernel-level system dump remains large enough to compromise availability. An even smaller memory dump may be acquired to cover only the smallest amount of base-level debugging information, typically sufficient only to identify a problem.

SUMMARY

[0006] In accordance with an embodiment of a computing system, a plurality of redundant, loosely-coupled processor elements are operational as a logical processor. A logic detects a halt condition of the logical processor and, in response to the halt condition, reintegrates and commences operation in less than all of the processor elements leaving at least one processor element nonoperational. The logic also buffers data from the nonoperational processor element in the reloaded operational processor elements and writes the buffered data to storage for analysis.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] Embodiments of the invention relating to both structure and method of operation, may best be understood by referring to the following description and accompanying drawings whereby:

[0008] **FIG. 1** is a schematic block diagram depicting an embodiment of a computing system that includes a plurality of redundant, loosely-coupled processor elements arranged and operational as a logical processor;

[0009] **FIG. 2** is a flow chart that illustrates an embodiment of a sequence of operations for performing an asymmetric memory dump operation;

[0010] **FIG. 3** is a flow chart showing an alternative embodiment of a method for performing a diagnostic memory dump operation;

[0011] **FIGS. 4A, 4B**, and **4**C are schematic block diagrams respectively showing an embodiment of a computer system;

[0012] **FIG. 5** is a schematic block diagram depicting another embodiment of a synchronization unit;

[0013] **FIG. 6** is a schematic block diagram showing a functional view of three processor slices operating in duplex mode with one processor slice omitted from running the operating system;

[0014] **FIG. 7** is a block timing diagram that illustrates an embodiment of the technique for performing a diagnostic memory dump;

[0015] **FIG. 8** is a schematic block diagram illustrating an embodiment of a processor complex in a processor node;

[0016] **FIG. 9** is a schematic block diagram showing an embodiment of a processor complex that includes three processor slices; and

[0017] **FIGS. 10A and 10B** are schematic block diagrams depicting an embodiment of a processor complex and logical processor.

DETAILED DESCRIPTION

[0018] Various techniques may be used to capture a memory dump of a computing system. In one example, a computing system composed of multiple redundant processors can capture a memory dump by having a central processing unit, for example running an operating system that supports multiple redundant processors such as the NonStop Kernel™ made available by Hewlett Packard Company of Palo Alto, Calif., copy the memory of a non-executing central processing unit. In a particular example, the non-executing or "down" processor can run a Halted State Services (HSS) operating system. The executing processor copies the memory dump data from the non-executing processor back to the executing processor's memory and subsequently writes the data to a storage device, such as a disk file.

[0019] In a specific example, a pre-Fast-Memory-Dump (pre-FMD) technique can be used in which the running processor copies raw data from the down processor memory and compresses the raw data in the running processor. In an alternative post-FMD method, the raw data is compressed in the down processor, for example under HSS, and the com-

pressed data is moved via network communications, such as ServerNet™ to the running processor which writes the compressed data to storage, such as a memory dump disk. A further alternative Fast-Memory-Dump enhancement involves copying only part of the memory, either compressed or noncompressed data, from the down processor to the running processor, then reloading the operating environment to the memory part to begin execution, then copying the remainder of the memory after completion of the reload that returns the memory to the running system.

[**0020**] The described dump techniques use capture of memory contents prior to reloading the processor, for example by copying the memory to the system swap files or special dump files. The copying time may be mitigated, for example by copying part of memory and reloading into only that copied part, then recopying the remainder of the data after the processor is reloaded, returning memory to normal usage after the copy is complete. The method reduces the time expenditure, and thus system down-time, for capturing and storing the memory dump data, but significantly impacts processor performance since only a subset of the memory is available for normal operation. The techniques further involve a significant delay before normal operation can be started.

[**0021**] In an illustrative example embodiment, a logical processor may include two or three processor elements running the same logical instruction stream. A dual-modular redundant (DMR) logical processor and/or a tri-modular redundant (TMR) logical processor can capture and save a logical processor memory dump while concurrently running an operating system on the sample logical processor. In a fully parallel technique that begins following a computing system halt condition, less than all of the processor elements are reloaded and made operational, leaving at least one processor element in a non-running or "down" condition, not integrated into the logical processor. Memory dump data is copied from the down processor element, for example using a dissimilar data exchange direct memory access (DMA) transfer.

[**0022**] Referring to **FIG. 1**, a schematic block diagram depicts an embodiment of a computing system **100** that includes a plurality of redundant, loosely-coupled processor elements **102** that are arranged and operational as a logical processor. A logic, for example executable in the processor elements **102**, detects a halt condition of the logical processor and, in response to the halt condition, reloads and commences operation in less than all of the processor elements, leaving at least one processor element nonoperational. The logic also buffers data from the nonoperational processor element in the reloaded operational processor elements and writes the buffered data to storage for analysis.

[**0023**] The loosely-coupled processors **102** form a combined system of multiple logical processors, with the individual processors **102** assuring data integrity of a computation. The illustrative computing system **100** operates as a node that can be connected to a network **104**. One example of a suitable network **104** is a dual-fabric Hewlett Packard ServerNet cluster™. The computing system **100** is typically configured for fault isolation, small fault domains, and a massively parallel architecture. The computing system **100** is a triplexed server which maintains no single point of hardware failure, even under processor failure conditions. The illustrative configuration has three instances of a processor slice **106**A, **106**B, **106**C connected to four logical synchronization units **108**. A logical synchronization unit **108** is shown connected to two network fabrics **104**.

[**0024**] Individual logical computations in a logical processor are executed separately three times in the three physical processors. Individual copies of computation results eventually produce an output message for input/output or interprocess communication, which is forwarded to a logical synchronization unit **108** and mutually checked for agreement. If any one of the three copies of the output message is different from the others, the differing copy of the computation is "voted out" of future computations, and computations on the remaining instances of the logical processor continue. Accordingly, even after a processor failure, no single point of failure proceeds to further computations. At a convenient time, the errant processing element can be replaced online, reintegrating with the remaining processing elements, restoring the computing system **100** to fully triplexed computation.

[**0025**] In the illustrative configuration, the individual processor slices **106**A, **106**B, **106**C are each associated respectively with a memory **110**A, **110**B, **110**C and a reintegration element **112**A, **112**B, **112**C. The data can be buffered or temporarily stored, typically in the memory associated with the running processor slices. The individual processor slices **106**A, **106**B, **106**C also can include a logic, for example a program capable of execution on the processor elements **102** or other type of control logic, that reloads and reintegrates the nonoperational processor element or slice into the logical processor after the data is buffered.

[**0026**] An asymmetric data dump is desirable to enable analysis into causes of a failure condition. Diagnostic information is more likely to be collected if acquisition can be made without compromising performance and availability.

[**0027**] The computing system **100** can activate capture of diagnostic memory dump information upon a logical processor halt condition. Diagnostic memory dump collection logic reloads the logical processor but does not include all processor slices **106**A, **106**B, **106**C in the reload. In some implementations and in some conditions, the processor slice selected to omit from reload is arbitrarily selected. In other implementations or conditions, the omitted processor slice can be selected based on particular criteria, such as measured performance of the individual slices, variation in capability and/or functionality of the particular processor slices, or the like. The processor slice omitted from the reload is maintained in the stopped condition with network traffic neither allowed into the stopped processor elements **102** nor allowed as output.

[**0028**] No matter the reason that the processor element is halted, the element may remain stopped and the associated memory may be dumped prior to reintegrating the element.

[**0029**] Referring to **FIG. 2**, a flow chart illustrates an embodiment of a sequence of operations for performing an asymmetric memory dump operation **200**. Upon a halt condition **202** of a logical processor, a response logic or program reloads the operating system **204** in less than all of the processor slices. In a specific example, one processor slice is omitted from reload while the operating system is reloaded in the other two processor slices. One technique for

reloading less than all processor slices is performed by issuing a command to place the logical processor in a "ready for reload" state that denotes which processor slice and/or processor element is to be omitted from the reload. The omitted processor slice or processor element is voted out and the remaining two processor elements in a tri-modular redundant (TMR) logical processor, or the remaining single processor element in a dual-modular redundant (DMR) logical processor, are reloaded. Criteria for selection of the omitted processor may be arbitrary or based on various conditions and circumstances.

[**0030**] After completing the partial reload, a parallel receive dump (PRD) program is automatically initiated **206**. A separate instruction stream of the PRD program typically executes in all reloaded processor slices, although other implementations may execute from logic located in alternative locations.

[**0031**] The parallel receive dump (PRD) program creates **208** a dump file and allocates **210** buffers; typically memory associated with the processor slices. All architectural state of the processor elements is typically saved in the memory so that separate collection of the information is superfluous.

[**0032**] The parallel receive dump (PRD) program opens **212** a memory window on a physical partition of memory associated with the particular processor slice executing the PRD program. The parallel receive dump (PRD) program moves **214** the window over the entire partition. In the illustrative embodiment, all processor elements of the logical processor generally have identical partitions so that the window also describes the partition of the omitted processor element.

[**0033**] The parallel receive dump (PRD) program performs a divergent data direct memory access (DMA) **216** operation that transfers data from one processor slice to another via DMA. A specific embodiment can use high order address bits of the source memory address to identify the particular omitted processor slice and/or processor element. For example, four memory views are available using ServerNet™ including the logical processor as a whole, processor slice A, processor slice B, and processor slice C. A direct memory access device reads data from the omitted processor slice and copies the data to a buffer in memory of the two executing processor slices, for example running the NonStop Kernel™.

[**0034**] In a particular implementation of the DMA operation, the source physical address high-order bits denote the one specific processor slice omitted from the reload. The target buffer for the DMA operation is in memory associated with the reloaded processor slices running the operating system and not the memory associated with the stopped processor slice.

[**0035**] The computing system that executes the asymmetric memory dump operation **200** may further include a reintegration logic that restarts and resynchronizes the plurality of processor elements following a failure or service condition. A reintegration process is executable on at least one of the operating processor elements and delays reintegration of the nonoperational processor element until dump processing is complete.

[**0036**] When the input/output operations for transferring the memory dump to buffers in the executing processor slices are complete, the parallel receive dump program compresses **218** the data into a compressed dump format, and writes **220** the compressed data to a storage device, for example a dump file on an external disk storage. Transfer of the compressed dump data is similar to the dump operation of the pre-Fast-Memory-Dump (pre-FMD) technique receive dump operation.

[**0037**] When the parallel receive dump program has completed copying, compressing, and writing the dump data to storage, the parallel receive dump program closes **222** the window to physical memory, closes **224** the storage file, and initiates **226** reintegration of the dumped processor slice.

[**0038**] The omitted processor slice is reintegrated **228** and the dump operation completes.

[**0039**] Referring to **FIG. 3**, a flow chart shows an alternative embodiment of a method for performing a diagnostic memory dump operation **300**. When a processor halts, a diagnostic dump of processor memory contents can be taken to determine the cause of the halt condition. A processor that includes multiple redundant loosely-coupled processor elements is called a logical processor. When a logical processor halts, the diagnostic dump can be taken without delaying the reloading —the return to service—of the logical processor, for example by reloading all but one of the multiple loosely-synchronized processor elements, leaving the one processor unchanged or "down" until the logical processor is reloaded.

[**0040**] The diagnostic memory dump operation **300** is invoked as a result of the halt condition. The condition causes one of the logical processors to halt while the other logical processors continue to run and do not require reloading. Data from the halted logical processor is dumped to one of the running logical processors, and the dump data is written to storage, such as a disk. The halted processor is then reloaded. Only one logical processor is halted and only one reload takes place. In a reload of a logical processor, a new copy of the operating system is brought up in the logical processor. For reintegration of a processor element, a processor is merged back into a running logical processor.

[**0041**] The diagnostic memory dump method **300** can be implemented in an executable logic such as a computer program or other operational code that executes in the processor elements or in other control elements. The operation begins on detection **302** of a halt condition of at least one processor element of multiple redundant processor elements. In a particular example, a system may implement a pointer to a linked list of various fault and operating conditions that causes execution to begin at a servicing logic for a particular condition. Various conditions may evoke a response that generates a diagnostic memory dump. One processor element, termed a "down" processor element, is maintained **304** in a state existing at the halt condition and the other processor elements are reloaded **306** and enabled to commence execution, for example restarting an operating system such as Hewlett Packard's NonStop Kemel™. One technique for initiating a response to the halt condition issues a command to place the logical processor in a "ready for reload " state in which the command designates the processor element to be omitted from the reload. The command causes "voting-out" of the omitted processor element and reloads the remaining processor elements for execution.

[**0042**] The state of the down processor maintained at the halt condition is copied **308** to a storage while the reloaded

other processors continue executing. For example, once reloading of the processor elements other than the down processor element is complete, the reloaded processor elements can automatically initiate a parallel receive dump program that creates a dump file, allocates buffers in memory associated with the reloaded processors for usage in temporarily storing the memory dump data, and saves the architectural state of all processor elements in memory buffers. In some embodiments, a special direct memory access (DMA) operation can be started that copies memory of the down processor element to buffers in the running processor elements. The divergent data Direct Memory Access (DMA) operation uses a self-directed write whereby a designated source memory address identifies the one processor element maintained in the halt condition.

[**0043**] On completion of the Input/Output operation of the divergent DMA operation, the parallel receive dump program compresses data into dump format, writes **312** the compressed memory dump data to a storage device, and closes the memory window. The divergent data DMA operation can be used to write **312** the memory dump from the buffers to a storage device, for example an external disk storage device for subsequent analysis.

[**0044**] After the data dump is copied from the down processor to the executing processors **308** and possibly concurrent with memory dump is transfer from the down processor to storage, such as the dump file and/or temporary buffer, the down processor element is reintegrated **310** into the logical processor.

[**0045**] The illustrative technique enables the logical processor to immediately return to service following a halt condition, eliminating or avoiding delay otherwise involved for copying of the diagnostic dump memory to disk. The illustrative technique further enables functionality as a highly available system, eliminating or avoiding removal of a logical processor from service for collection of diagnostic information which may take a substantial amount of time. Commonly the transferring of the data to disk can take several minutes, a large amount of time in a highly-available system. The illustrative technique reduces the latency effectively to zero.

[**0046**] Referring to **FIGS. 4A, 4B**, and **4**C, schematic block diagrams respectively show an embodiment of a computer system **400**, for example a fault-tolerant NonStop™ architecture computer system from Hewlett-Packard Company of Palo Alto, Calif., and two views of an individual processor slice **402**. The illustrative processor slice **402** is an N-way computer with dedicated memory and clock oscillator. The processor slice **402** has multiple microprocessors **404**, a memory controller/IO interface **408**, and a memory subsystem **406**. The processor slice **402** further includes reintegration logic **410** and an interface to voter logic.

[**0047**] In the event of a halt condition, a logical processor halts. The computer system **400** is an entire process made up of multiple logical processors. The halted logical processor ceases functioning under the normal operating system, for example the NonStop Kernel system, and enters a state that allows only abbreviated functionality. In one example, the halt condition causes the system to function under halted state services (HSS). A failure monitoring logic, such as software or firmware operating in control logic, detects the halt condition, and selects from among the processor slices **402** for a processor slice to omit from reloading. In various implementations the omitted processor may be selected arbitrarily or based on functionality or operating characteristics, such as performance capability considerations associated with the different processor slices **402**. The control logic reloads the operating system into memory for processor slices that are not selected for omission so that the processor slices return to execution. The omitted processor slice remains inactive or "down", continuing operations under halted state services (HSS).

[**0048**] The reloaded processor slices request capture of a memory dump from the omitted processor memory while the omitted processor slice remains functionally isolated from the operating processor slices. The reloaded processors begin a copy process, for example that executes on the reloaded processors and stores the memory dump data from the omitted processor to memory **406** associated with the operating processor slices **402**. The diagnostic dump data passes from the omitted processor slice to the reloaded and operating processor slices via a pathway through a logical synchronization unit **414**. The logical synchronization unit **414** is an input/output interface and synchronization unit that can be operated to extract the memory dump data from the omitted processor slice including a copy of the data and a copy of a control descriptor associated with the data. The reintegration logic **410** generally operates in conditions of processor slice failure to reintegrate the operations of the failed slice into the group of redundant slices, for example by replicating write operations of memory for one processor slice to memory of other processor slices in a redundant combination of processor slices.

[**0049**] In a specific example, software executing in one or more of the reloaded and running processor slices performs asymmetric input/output operations that copy memory dump data from the omitted processor slice to memory buffers in both of the operating, reloaded processor slices. Accordingly, the two reloaded processor slices **402** operate temporarily in duplex mode while acquiring and storing the diagnostic dump information before returning to triplex operation.

[**0050**] In a particular embodiment, the microprocessors **404** may be standard Intel Itanium Processor Family multiprocessors that share a partitioned memory system. Each microprocessor may have one or more cores per die. A processor slice **402** with an N-Way Symmetrical Multi-Processor (SMP) supports N logical processors. Each logical processor has an individual system image and does not share memory with any other processor.

[**0051**] Reintegration logic **410** can replicate memory write operations to the local memory and sends the operations across a reintegration link **412** to another slice. The reintegration logic **410** is configurable to accept memory write operations from the reintegration link **412**. The reintegration logic **410** can be interfaced between the I/O bridge/memory controller **408** and memory **406**, for example Dual In-line Memory Modules (DIMMs). Alternatively, the reintegration logic **410** may be integrated into the I/O bridge/memory controller **408**. Reintegration logic **410** is used to bring a new processor slice **402** online by bringing memory state in line with other processor slices.

[**0052**] In an illustrative example, the computer system **400** uses loosely lock-stepped multiprocessor boxes called

slices **402**, each a fully functional computer with a combination of microprocessors **404**, cache, memory **406**, and interfacing **408** to input/output lines. All output paths from the multiprocessor slices **402** are compared for data integrity. A failure in one slice **402** is transparently handled by continuing operation with other slices **402** continuing in operation. The computer system **400** executes in a "loose-lock stepping" manner in which redundant microprocessors **404** run the same instruction stream and compare results intermittently, not on a cycle-by-cycle basis, but rather when the processor slice **402** performs an output operation. Loose-lockstep operation prevents error recovery routines and minor non-determinism conditions in the microprocessor **404** from causing lock-step comparison errors.

[**0053**] Referring to **FIG. 5**, a schematic block diagram depicts an embodiment of a synchronization unit **500** including a logical gateway **514** that prevents divergent operations from propagating to the input/output stream. The synchronization unit **500** is capable of connecting to one, two, or three processor slices **504** through a serialized input/output bus. The synchronization unit **500** performs transaction-level checking on input/output transactions and forwards data to a host-side input/output device **522**, for example a host bus adapter or storage array network (SAN) controller. The synchronization engine **520** enables the multiple processor slices **504** to synchronize and exchange asynchronous data such as interrupts and can also control exchange of private and dissimilar data among slices.

[**0054**] Logical gateway **514** has two independent voter subunits, one for voting Programmed Input/Output (PIO) read and write transactions **516** and a second for voting Direct Memory Access (DMA) read responses **518**. The Direct Memory Access (DMA) read response subunit **518** verifies input/output controller-initiated DMA operations or responses from memory and performs checks on read data with processors performing voted-write operations. DMA write traffic is originated by the input/output controller **522** and is replicated to all participating processor slices **504**. DMA read traffic is originated by the input/output controller **522**. DMA read requests are replicated to all participating slices **504**.

[**0055**] Referring to **FIG. 6**, a schematic block diagram illustrates a functional view of three processor slices **600**A, **600**B, and **600**C. The logical gateway supports single slice sourced read transactions in which a read response of one processor slice alone is transferred from the processor, enabling performance of a memory dump operation. For example, the single slice sourced read can be used to transfer data in a pathway **608** from omitted processor slice memory **610** to memory **612** and **614** in the executing processor slices.

[**0056**] Referring to **FIG. 7**, a block timing diagram illustrates an embodiment of the technique for performing a diagnostic memory dump **700**. The computing system runs in a triplex mode **702** with the three individual processor slices executing a common instruction stream redundantly. A halt condition **704** terminates execution of the processor slices and entry of all slices into halted state services (HSS). A processor slice is selected to omit from running of the processor slices in duplex and the two processor slices not selected for omission are reloaded **706**. After reload, the two processor slices run the operating system in duplex **708**. After reload is complete and the processor slices run in duplex, a copy process starts **710**, for example in a logical synchronization unit that operates as a pathway for copying data from the omitted processor slice memory to a buffer memory in one or both of the running processor slices under management of control information received from the running processor slices. Data can be copied through a scan of the entire memory partition of the omitted processor slice memory. When the diagnostic dump data copy is completed **712**, reintegration **714** of the omitted processor slice with the two running processor slices begins, enabling upon completion of integration triplex mode operation **716** of the three processor slices.

[**0057**] Referring to **FIG. 8**, a schematic block diagram illustrates an embodiment of a processor complex **802** in a processor node **800**. A computer system runs a single application on multiple logical processors, each executing the same instruction stream by different microprocessors on different processor boards. The individual processor complexes **802** implement multiple logical processors. An illustrative processor node **800** has three processor complexes **802**, each with four logical processors so that the entire node **800** has twelve logical processors. In an example implementation inter-processor communication and processor-to-IO communication may use redundant System Area Networks (SANs) **804**. Individual logical processors in the processor complex **802** have interfaces **806** to the SANs **804**. Logical processors use the SANs **804** for interprocessor communication and connectivity with shared input/output (I/O) adapters **808**.

[**0058**] Referring to **FIG. 9**, a schematic block diagram illustrates an embodiment of a processor complex **900** that includes three processor slices, slice A **902**, slice B **904**, and slice C **906**, and N voting blocks **908** and System Area Network (SAN) interfaces **910**. The number of voting blocks N is the number of logical processors supported in the processor complex **900**. A processor slice **902**, **904**, **906** is illustratively a multiprocessor computer with contained caches, a memory system, a clock oscillator, and the like. Each microprocessor is capable of running a different instruction stream from a different logical processor. The N voting blocks **908** and N SAN interfaces **910** are mutually paired and included within N respective logical synchronization units (LSUs) **912**. An illustrative processor complex **900** has one to two logical synchronization blocks **912**, with associated voting block unit **908** and SAN interface **910**, per logical processor.

[**0059**] During operation, the processor slices A **902**, B **904**, and C **906** generally are configured as multiple tri-modular logical processors that execute in loose lockstep with I/O outputs compared by the voter units **908** before data is written to the System Area Network **914**.

[**0060**] The term processor complex **900** is merely descriptive term and does not necessarily define a system enclosed within a single housing. A processor complex is generally not a single field-replaceable unit. In a typical minimum configuration, a field-replaceable unit can include one LSU and one slice.

[**0061**] The voter units **908** are logical gateways of operation and data crossing from the logical synchronization blocks **912** unchecked domain to a self-checked domain. DMA read data and PIO reads and write requests that

address the self-checked domain are checked by the voter unit **908** in order of receipt. Operations are not allowed to pass one another and complete before the next is allowed to start. DMA read response data are also checked in the order received and then forwarded to the system area network interface **910**, for example a Peripheral Component Interconnect Extended (PCI-X) interface. PIO requests and DMA read responses are processed in parallel with no ordering forced or checked between the two streams.

[**0062**] Referring to **FIG. 4** in combination with **FIG. 9**, a processor complex includes reintegration links **412** that copy memory contents from a functioning slice or slicess to a non-operating or newly added slice. Reintegration is used after some errors or repair operations in conditions that recovery is served by resetting a slice and returning to operation with other running slices. In various embodiments, the reintegration link **412** may copy over the memory of a single processor element, multiple processor elements, or all processor elements within a processor slice.

[**0063**] Referring to **FIG. 10A**, a schematic block diagram depicts an embodiment of a processor complex **1000**. A processor element **1002** is defined herein as hardware that executes an individual instruction stream running on a single processor slice **1004**. The illustrative processor complex **1000** may be considered a matrix with three columns A, B, and C, representing three processor slices **1004** and N rows representing N logical processors. In the illustrative diagram, the notation PEA_{A3} refers to the processor element that is the third microprocessor in slice A. The processor element **1002** is one logical processor, for example executing one instruction stream, on one processor slice **1004**.

[**0064**] If the microprocessor used in the processor slice **1004** has multiple cores, for example the microprocessor die has multiple processor cores, each running an independent instruction stream. The term processor element refers to a single core.

[**0065**] Referring to **FIG. 10B**, a schematic block diagram illustrates the embodiment of the processor complex **1000**, depicting a logical processor **1006**. Within a processor slice **1004**, for example the N-way SMP processor slice, each instruction stream is associated to a different logical processor **1006**. Each logical processor **1006** can execute a processor-dedicated copy of an operating system, for example the NonStop Kernel (NSK)™ operating system from Hewlett-Packard Company of Palo Alto, Calif. Within the processor slice **1004** are N logical processors **1006** that mutually share neither private memory nor peripheral storage, but otherwise all run out of the same physically-shared memory. Except for a small amount of initialization code that segments the processor slice memory, each logical processor runs independently of the others from different regions of the same memory.

[**0066**] The logical processor **1006** is formed from one or more processor elements **1002**, for example three in the illustrative embodiment, depending on the number of processor slices **1004** available. A simplex logical processor has only one processor element (PE) per logical processor. A dual-modular redundant (DMR) logical processor has two processor elements (PEs) per logical processor. A tri-modular redundant (TMR) logical processor has three. Each processor element **1002** in a logical processor **1006** runs the same instruction stream, in loosely lock-stepped operation, and output data from multiple processor elements is compared during data input/output (I/O) operations.

[**0067**] Referring again to **FIG. 5** in combination with **FIGS. 9, 10A**, and **10**B, the logical synchronization unit (LSU) **500** functions as part of a logical processor **1006** in a fault tolerant interface to a system area network **914** and performs voting and synchronization of the processor elements **1002** of the logical processor **1006**. In an illustrative implementation, each logical synchronization unit **500** is controlled and used by only a single logical processor **1006**.

[**0068**] In the illustrative embodiment, one or two logical synchronization units **500** are combined with one, two, or three processor elements **1002** to create varying degrees of fault tolerance in the logical processors **1006**. A system may optionally be configured with a second logical synchronization unit **500** per logical processor **1006**.

[**0069**] The logical synchronization unit **500** may use fully self-checked logic. The logical synchronization unit **500** resides between the replicated processor slices **902**, **904**, **906** and the system area network **914** and, in some implementations, may not have data compared to preserve data integrity such as for the processor slices. Accordingly, appropriate redundancy techniques may be used in the logical synchronization unit **500** to ensure data integrity and fault isolation.

[**0070**] The voter logic **908** connects the processor slices **902**, **904**, **906** to the SAN interface **910** and supplies synchronization functionality for the logical processor. More specifically, the voter logic **908** compares data from programmed input/output (PIO) reads and writes to registers in the logical synchronization unit **912** from each of the processor elements. The comparison is called voting and ensures that only correct commands are sent to logical synchronization unit logic. Voter logic **908** also reads outbound data from processor slice memories and compares the results before sending the data to the system area network (SAN), ensuring that outbound SAN traffic only contains data computed, or agreed-upon by voting, by all processor elements in the logical processor. The voter logic **908** also replicates and distributes programmed input/output (PIO) data read from the system area network and registers in the logical synchronization unit **912** to each of the processor elements. The voter logic **908** further replicates and distributes inbound data from the system area network to each of the processor elements. The voter logic **908** can supply time-of-day support to enable processor slices to simultaneously read the same time-of-day value. The voter logic **908** supports a rendezvous operation so that all processor slices can periodically check for mutual synchrony, and cause one or more processor elements to wait to attain synchrony. The voter logic **908** also supports asymmetric data exchange buffers and inter-slice interrupt capability.

[**0071**] The voter logic, shown as the logical gateway **514** in **FIG. 5**, includes interface logic, for example programmed input/output (PIO) **516** and direct memory access (DMA) read interface **518**, and state logic. The state logic designates either an asymmetric state or a symmetric state. The asymmetric state is specific to one processor element **504**. The symmetric state is common to the entire logical processor. Examples of processor element-specific logic and data are the rendezvous registers and logic shown as synchronization engine **520**, dissimilar data exchange buffers, and the inter-

slice interrupts. Parallel read and write operations to the asymmetric logic are from a single processor element **504**. Processor element-initiated read and write operations to the asynchronous registers are not voted or compared. Data is sent back only to the specific processor element requesting the operation.

[**0072**] The logical gateway **514** forwards data to the processor element memories **506** at approximately the same time. However, the processor elements **504** do not execute in perfect lockstep so that data may arrive in memory **506** early or late relative to program execution of the particular processor element **504**. No data divergence results among the processor elements since the SAN programming model does not allow access to an inbound buffer until after arrival notification receipt.

[**0073**] The system area network (SAN) interface **522** is generally used for all input/output, storage, and interprocessor communications. The SAN interface **522** communicates with the three processor slices through the logical gateway **514**. System area network (SAN) traffic passes to and from a logical processor and not individual processor elements. The logical gateway **514** replicates data from the system area network to the memory of all processor elements **504** participating in the logical processor. The logical gateway **514** also performs the voting operation, comparing data from the slices before passing the data to the SAN interface.

[**0074**] In an illustrative configuration, each logical processor has a dedicated SAN interface. To avoid failures or bugs that can affect multiple logical processors, redundant execution paths can be implemented to avoid a single failure from disabling multiple logical processors.

[**0075**] The system performs according to a loose lock-step fault tolerance model that enables high availability. The system is tolerant of hardware and many software faults via loosely-coupled clustering software that can shift workload from a failing processor to the other processors in the cluster. The model tolerates single hardware faults as well as software faults that affect only a single processor. The model uses processor self-checking and immediately stopping before faulty data is written to persistent storage or propagated to other processors.

[**0076**] After a failure or a service operation on a multiple-slice system the new processor or processors are reintegrated, including restarting and resynchronizing with the existing running processors. Steps to restore the processor memory state and return to loose lock-step operation with the existing processor or processors is called reintegration. Unlike a logical processor failure, multiple-redundant hardware failures and the subsequent reintegration of the replacement or restarted hardware are not detectable to application software. Reintegration is an application-transparent action that incorporates an additional processor slice into one or more redundant slices in an operating logical processor.

[**0077**] In the reintegration action, all memory and processor state of a running slice is copied to the second slice and both slices continue operation. The basic approach is complicated because reintegration is to have minimal duration, performance, and availability impact on the running processor, for example the reintegration source that continues to execute application programs.

[**0078**] In some embodiments, special hardware called a reintegration link is used to copy memory state from the reintegration source to the target. Since the processor on the reintegration source continues executing application code and updating memory, the reintegration link allows normal operations to modify memory and still have modifications reflected to the reintegration target memory.

[**0079**] Reintegration link hardware can be implemented so that reintegration occurs in groups of one or more logical processors. For example, an implementation can reintegrate an entire slice even if only one processor element of one logical processor is restarted. A reintegration scheme that affects only a single logical processor reduces or minimizes the amount of time a system runs on less than full capability.

[**0080**] Reintegration is triggered by a condition such as a processor slice replacement, receipt of a command for a system management facility, and/or occurrence of an input/output voting error or other error detected by other processor elements in the logical processor. For processor slice replacement, each new processor elements is reintegrated by the currently running logical processors. For a detected error, remaining executing processor elements can reset the faulty processor element and reintegrate. For a transient error, the processor element can be brought back to a fully functional state.

[**0081**] The logical processor that can reintegrate a new processor element determines whether to begin the reintegration process. For reintegration of an entire slice, control resides in the processor complex. In both cases reintegration control is below the system level function. If reintegration fails, the logical processor simply logs the error and continues to attempt the reintegration process. The logical processor may reduce the frequency of reintegration, but continues to try until success.

[**0082**] Referring again to **FIGS. 4A, 4B**, and **4**C, reintegration control is performed by the reintegration logic **410** that is positioned between the processors **404** and memory **406**. A reintegration link **412** connects the multiple processor slices **402** and can reflect memory writes from one slice to an adjacent neighbor slice. In an illustrative embodiment, the reintegration logic **410** can be a double data rate synchronous dynamic random access memory (DRAM) interface. The reintegration logic **410** performs multiple functions.

[**0083**] During normal operation the reintegration logic **410** transparently passes memory operations between the microprocessors **404** and local memory **406**. Reintegration link **412** usage is generally limited to scrubbing of latent faults.

[**0084**] During the reintegration operation, reintegration logic **410** at the source processor slice duplicates all main memory write operations, sending the operation both to the local memory **406** and across the reintegration link **412**. At the target processor slice, reintegration logic **410** accepts incoming writes from the reintegration link **412** and writes target local memory **406**. During reintegration the target does not execute application programs but rather executes a tight cache resident loop with no reads or writes to target local memory **406**.

[**0085**] The reintegration link **412** is a one-way connection from one processor slice to one adjacent neighbor processor

slice. For a system that includes three processor slices A, B, and C, only slice A can reintegrate slice B, only slice B can reintegrate slice C, and only slice C can reintegrate slice A. Starting from one processor slice with two new processor slices, reintegration can be done in two steps. The first reintegration cycle brings the second processor slice online and the second reintegration cycle brings the third online.

[**0086**] For reintegration of a single processor slice, reintegration logic **410** on the source and target slices are initialized. The logical synchronization unit **414** is set unresponsive to the target, no interrupts are delivered to the target and input/output operations do not include the target. The target is set to accept writes from the reintegration link **412**. The target executes an in-cache loop waiting for reintegration to complete. While maintaining local processing, the source reads and writes back all memory local to the source, in an atomic operation since the SAN interface may simultaneously be updating target source memory. A single pass operation reads each cache block from memory and then tags the cache block as dirty with an atomic operation without changing contents. Then, target memory is updated except for the state contained in the remaining dirty cache blocks on the source cache. All processes are suspended and architectural state, for example the processor registers, are stored. Caches are flushed and can be validated. Cache validation is optional but facilitates synchronization of the reintegration source and target after completion of reintegration. Without validation, the source would consistently have a cache hit while the target would miss. All writes are reflected to the target processor, reintegrating the final bytes of dirty cache data into the target processor slice. The logical synchronization unit **414** is enabled to read and write to the target, stop write reflecting, spin in a cache loop, and perform the rendezvous operation. After the rendezvous operation completes, the target should have the exact same state as the source. Architectural state is restored from memory **406**, and operation resumes.

[**0087**] In the illustrative implementation, reintegration affects all processor elements in a slice. Alternatively stated, even if a failure in a triple-redundant processor system affects only one processor element of one logical processor, reintegration is performed to the entire memory of the affected processor slice. In the triple-redundant system, the processor complex executes applications with two processor slices active during reintegration so that no loss of data integrity occurs.

[**0088**] In the illustrative implementation, reintegration is performed on an entire processor slice. In other implementations, a single processor element of the processor slice may be reintegrated.

[**0089**] Reintegration enables hardware to recover from a hardware fault without application acknowledgement of the failure. An alternative to reintegration may be to halt the logical processor and allow the application workload to shift to other logical processors. The halted processor can then be restarted, effectively reintegrating the failing processor element in a logical processor.

[**0090**] While the present disclosure describes various embodiments, these embodiments are to be understood as illustrative and do not limit the claim scope. Many variations, modifications, additions and improvements of the described embodiments are possible. For example, those having ordinary skill in the art will readily implement the steps necessary to provide the structures and methods disclosed herein, and will understand that the process parameters, materials, and dimensions are given by way of example only. The parameters, materials, components, and dimensions can be varied to achieve the desired structure as well as modifications, which are within the scope of the claims. Variations and modifications of the embodiments disclosed herein may also be made while remaining within the scope of the following claims. For example, the specific embodiments described herein identify various computing architectures, communication technologies and configurations, bus connections, and the like. The various embodiments described herein have multiple aspects and components. These aspects and components may be implemented individually or in combination in various embodiments and applications. Accordingly, each claim is to be considered individually and not to include aspects or limitations that are outside the wording of the claim.

What is claimed is:

1. A method for performing a diagnostic memory dump comprising:

detecting a halt condition of at least one processor element of multiple redundant processor elements;

maintaining one processor element in a state existing at the halt condition;

reloading others of the processor elements whereby the others commence execution;

copying the state of the maintained one processor element to a storage while the others continue executing; and

reintegrating the one processor element when the halt condition copying is complete whereby the one processor element commences execution.

2. The method according to claim 1 wherein:

the multiple redundant processor elements are loosely-synchronized processor elements.

3. The method according to claim 1 further comprising:

copying memory of the maintained one processor element to buffers in the executing others of the processor elements using a Direct Memory Access (DMA) operation; and

subsequently writing the copied memory to a storage device for later analysis.

4. The method according to claim 1 further comprising:

initiating a response to a halt condition of a logical processor that comprises a plurality of redundant processor elements;

issuing a command to place the logical processor in a "ready for reload" state, the command designating a processor element that is omitted from reintegration; and

executing the command further comprising:

voting-out the designated omitted processor element; and

reintegrating remaining processor elements.

5. The method according to claim 1 further comprising:

detecting completion of reintegrating the others of the processor elements; and

automatically initiating a parallel receive dump program; and

executing the parallel receive dump program further comprising:

creating a dump file;

allocating buffers; and

saving architectural state of all processor elements in memory.

6. The method according to claim 5 wherein executing the parallel receive dump program further comprises:

opening a memory window on a memory physical partition; and

moving the window over the partition.

7. The method according to claim 5 wherein executing the parallel receive dump program further comprises:

executing a divergent data Direct Memory Access (DMA) operation with a self-directed write whereby a designated source memory address identifies the one processor element maintained in the halt condition.

8. The method according to claim 6 wherein executing the parallel receive dump program further comprises:

upon completion of the Input/Output operation of the divergent DMA operation, compressing data into dump format and writing the compressed data to a dump file; and

closing the memory window.

9. A computing system comprising:

a plurality of redundant, loosely-coupled processor elements operational as a logical processor; and

a logic that detects a halt condition of the logical processor and, in response to the halt condition, reintegrates and commences operation in less than all of the processor elements leaving at least one processor element nonoperational, buffers data from the nonoperational processor element in the reintegrated operational processor elements, and writes the buffered data to storage for analysis.

10. The computing system according to claim 9 further comprising:

a logic that reloads and reintegrates the nonoperational processor element into the logical processor after the data is buffered.

11. The computing system according to claim 9 further comprising:

a parallel receive dump program that is initiated after reload and commencement of the operational processor elements, the parallel receive dump program that creates a dump file and allocates the buffers, and saves architectural state of the processor elements in a memory.

12. The computing system according to claim 11 further comprising:

the parallel receive dump program that opens a memory window on a physical partition allocated to a processor element upon which the parallel receive dump program executes, and that moves the memory window over the entire physical partition.

13. The computing system according to claim 12 further comprising:

a direct memory access device coupled to the processor element plurality; and

the parallel receive dump program that executes a divergent data input/output operation using the direct memory access device that identifies the nonoperational processor element and transfers data from the nonoperational processor element to at least one operational processor element by direct memory access.

14. The computing system according to claim 13 wherein:

the parallel receive dump program identifies the nonoperational processor element using address bits.

15. The computing system according to claim 13 further comprising:

an engine that generates data transfer request packets and tracks associated response packets;

an access validation and translation module that verifies legitimacy of incoming data transfer packets and translates addresses of legitimate packets into internal memory space; and

the parallel receive dump program that holds block transfer engine descriptors and access validation and translation tables in the operational processor elements for managing the direct memory access operation.

16. The computing system according to claim 13 further comprising:

a reintegration logic that restarts and resynchronizes the plurality of processor elements following a failure or service condition; and

a reintegration process executable on at least one of the operating processor elements that delays reintegrating the nonoperational processor element until dump processing is complete.

17. The computing system according to claim 13 wherein:

the parallel receive dump program determines when data transfer from the nonoperational processor element to the at least one operational processor element is complete and, upon completion, compresses the transferred data into a dump format and writes the compressed data to a dump file.

18. The computing system according to claim 17 wherein:

the parallel receive dump program determines when writing of the compressed data to the dump file is complete and, upon completion, closes the window to physical memory, closes the dump file, and initiates reintegration of the nonoperational processor element.

19. An interface for usage in a redundant processor comprising:

a direct memory access device coupled to a plurality of redundant, loosely-coupled processor elements operational as a logical processor; and

a data transfer program that executes a divergent data input/output operation using the direct memory access device that transfers data from one source processor element of the processor element plurality to at least one target processor element of others in the plurality of processor elements by direct memory access.

20. The interface according to claim 19 wherein:

the data transfer program identifies the source processor element using address bits.

21. The interface according to claim 19 further comprising:

an engine that generates data transfer request packets and tracks associated response packets;

an access validation and translation module that verifies legitimacy of incoming data transfer packets and translates addresses of legitimate packets into internal memory space; and

the data transfer program that holds block transfer engine descriptors and access validation and translation tables in the operational processor elements for managing the direct memory access operation.

22. The interface according to claim 19 further comprising:

a reintegration logic restarts and resynchronizes the plurality of processor elements following a failure or service condition; and

a reintegration process executable on at least one of the processor elements that delays reintegrating a selected nonoperational processor element.

23. A method for performing a diagnostic memory dump comprising:

detecting a halt condition of at least one processor element of multiple redundant processor elements;

maintaining one processor element in a state existing at the halt condition;

reloading others of the processor elements whereby the others commence execution;

copying the state and memory of the maintained one processor element, the state being copied to a storage while the others continue executing, the memory being copied to buffers in the executing others of the processor elements using a Direct Memory Access (DMA) operation; and

subsequently writing the copied memory to a storage device for later analysis.

24. The method according to claim 23 further comprising:

reintegrating the one processor element when the halt condition copying is complete whereby the one processor element commences execution.

* * * * *